Chapter 1: . . . And UNIX Lurks Beneath

In This Chapter

- **Why use UNIX?**
- **Doing things with the keyboard**
- **Introducing UNIX commands**
- **Creating text files**
- **Exploring deep inside Mac OS X**

As I mention in the first chapter of the book — at the beginning of our Lion odyssey — UNIX lurks deep beneath the shiny Aqua exterior of Mac OS X. UNIX is a tried-and-true operating system that's been around for decades, since the days when mainframe computers were king. If you don't believe that it's a powerful (and popular) operating system, consider that over half of all web servers on the Internet use some variety of UNIX as their operating system of choice.

Besides being battle-tested and having a long history, UNIX offers some fantastic features. Unlike the graphical world of Mac OS X, the keyboard plays an integral role in using a UNIX-based operating system. Because UNIX is text-based, you'll find that it's evolved a large set of useful keyboard-driven commands that can perform powerful feats that a mouse user just can't easily equal. This chapter examines the role of the keyboard in UNIX operating systems and describes how to execute standard file system commands. You also discover how to use Apple's additional set of commands and install your own commands (and simple programs) from the Internet.

COPYRIGHTED MATERIAL

Why Use the Keyboard?

To begin benefiting from the UNIX underpinnings of Mac OS X, get used to doing things with the keyboard. Although mouse skills can be applied to UNIX, you'll generally find performing UNIX functions faster and easier with the keyboard.

UNIX keyboarding is fast

Why on Earth would any red-blooded Macintosh owner want to leave the comfort of the mouse to use a keyboard? After all, the graphical user interface is what made the Macintosh great in the first place. With the Finder, you can navigate and manage the various files on your hard drive with a few clicks. This sounds simple enough, but for some tasks, using the keyboard can be just as fast, if not faster.

Suppose, for example, that you need to copy a file from somewhere on your hard drive to somewhere else on that same drive. To do so with the Finder, you must first open a Finder window (by clicking the Finder icon in the Dock or by double-clicking a drive icon on your Desktop). Then, by using a succession of mouse clicks, you navigate to where the file that you wish to copy resides. Next, you might open another Finder window and navigate to the folder where you wish to copy the file. (Note that opening the second Finder window requires pressing ⌘+N; clicking the Finder icon in the Dock doesn't open a second Finder window.) Finally, you duplicate the original file and drag that copy to its intended destination.

Comparatively, by using the keyboard and the power of UNIX, you can accomplish the same task with a one-line command. For some tasks, the mouse is definitely the way to go, but you can perform some other tasks just as quickly, if not faster, with the keyboard. For the skinny on one-line commands, skip down to the upcoming section "Uncovering the Terminal."

The UNIX keyboard is a powerful beast

So maybe you're not an expert typist, and using the mouse still sounds inviting. For many scenarios, you'd be correct in assuming that a mouse can handle the job just as quickly and easily as a bunch of commands that you have to memorize. Using the keyboard, however, offers some other distinct advantages over the mouse. To allow you to control your computer from the keyboard, all UNIX operating systems offer the *command-line tool.* With this tool, you can enter commands one line at a time: hence, its name. Mac OS X ships with the command-line application, Terminal. You can find it here:

```
/Applications/Utilities/Terminal
```

One shining feature of the command line is its efficiency. To wit: When you use a mouse, one mouse click is equal to one command. When you use the command line, on the other hand, you aren't limited to entering one command at a time; rather, you can combine commands into a kind of *supercommand* (minus the silly cape, but with bulging muscles intact), with each command performing some action of the combined whole. By using the command line, you can string together a whole bunch of commands to do a very complex task.

For example, consider how many times you'd have to click a mouse in the Finder to do the following:

1. Find all files that begin with the letters *MyDocument.*
2. From this list of files, add a number to the beginning of the filename, indicating its size in kilobytes.
3. Save the names of all altered files to a text file.

By using the command line, you could accomplish all these tasks by typing only one *super-command:* that is, a collection of three simple commands combined to form one instruction. The built-in Terminal program that ships with Mac OS X Lion gives you everything that you need to start using the command line. I show you how in the section "Uncovering the Terminal," later in this chapter.

Delving further into super-commands isn't for the faint-hearted; things get pretty ugly pretty quickly, and this chapter can only show you the very beginning of the UNIX Yellow Brick Road. Therefore, if your thirst for UNIX dominance so compels you, I invite you to do a little independent study to bone up on the operating system. Pick up a copy of the great book of lore entitled *UNIX For Dummies, 5th Edition*, written by John R. Levine and Margaret Levine Young (Wiley).

Go where no mouse has gone before

The Finder is generally a helpful thing, but it makes many assumptions about how you work. One of these assumptions is that you don't have any need to handle some of the files on your hard drive. As I mention in Book II, Chapter 6, Mac OS X ships with its system files marked Off Limits, and I generally agree with that policy (which keeps anyone from screwing up the delicate innards of Mac OS X). To secure your system files, Apple purposely hides some files from view.

But what road do you take if you actually need to view or modify those system files? Yep, you guessed it: The command line comes to the rescue! You can use the command line to peer inside every nook and cranny of your Mac's vast directory structure on your hard drive. It also has the power to edit files that aren't normally accessible to you. With the command line, you can pretend to be other users — even users with more permissions. By temporarily acting as another more powerful user, you can perform actions with the command line that would be impossible in the Finder. (Just remember to make sure that you know *exactly* what you're doing, or you're working with an Apple technical support person — a wrong move, and it'll be time for an Ominous Chord.)

Automate to elevate

If all these benefits are beginning to excite you, hold on to your socks! Not only can you perform complex commands with the command line, you can go even one step further: *automation.* If you find yourself using the same set of commands more than once, you're a likely candidate for using automation to save time. Instead of typing the list of commands each time, you can save them to a text file and execute the entire file with only one command. Now that's power, right up there with the dynamic duo of AppleScript and Automator! (Granted, it's not graphical like Automator, but then again UNIX has been around for decades.)

Of course, you probably don't like doing housekeeping tasks while you're busy on other things, so schedule that list of commands to run in the middle of the night while you're fast asleep. The command line lets you do that, too.

Note that automation of UNIX commands is totally separate from automation of Mac OS X applications with AppleScript and Automator, which I cover in Book VIII, Chapter 2.

Remote control

"So, Mark, the command line is the cat's meow for efficiently accessing and working with files on my Mac, and I can use it to automate many operations. Anything else?" I'm glad you asked! By using the command line, you can also send commands to another computer anywhere in the world (as long as you know the right login and password). After you log into another computer, you can use the same commands for the remote computer.

UNIX was created with multiple users in mind. Because computers used to be expensive (and honking *huge* machines to boot), UNIX was designed so that multiple users could remotely use the same machine simultaneously. In fact, if Mac OS X is your first encounter with UNIX, you might be surprised to know that many UNIX beginners of the past weren't even in the same room, building, state, or even country as the computer that they were using.

Not only can you work with a computer that's in a different physical location, but it's also very fast to do so. Instead of the bandwidth hog that is the Internet, the command line is lean and mean. This permits you to use a remote computer nearly as fast as if it were sitting on the desk in front of you. (This is a great advantage for road warriors who need to tweak a web or an e-mail server from a continent away.)

Uncovering the Terminal

The best way to find out how to use the command line is to jump right in. Mac OS X comes stocked with an application named *Terminal.* As I mentioned earlier, the Terminal application is where you enter commands in the command line, and it's located in the Utilities folder within the Applications folder on your hard drive — choose Applications➪Utilities (from the keyboard, press Shift+⌘+U).

Double-click the icon, as shown in Figure 1-1, to launch Terminal.

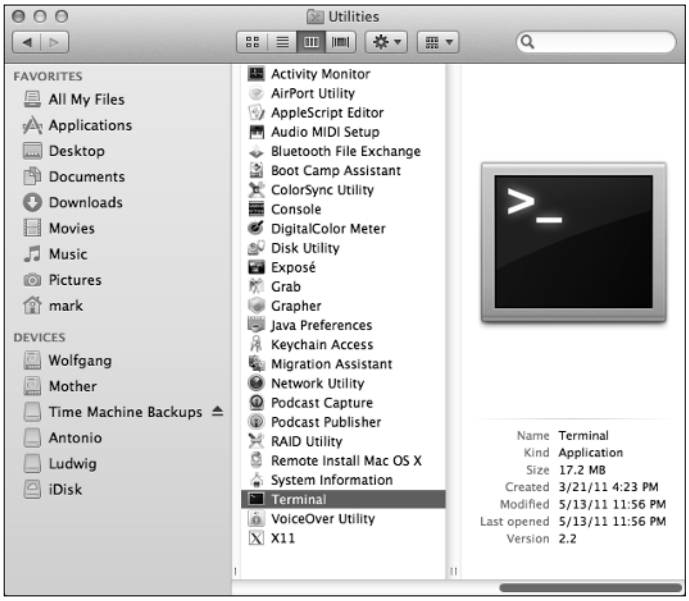

Figure 1-1: Find the Terminal application in your Utilities folder.

By the way, feel free to make Terminal more accessible by dragging its icon to the Dock or the toolbar. That way, you won't have to dive this deep into the Applications folder in the future.

What's a prompt?

Upon launch of the Terminal application, you'll immediately notice some text in the window that appears on-screen:

```
Last login: Sun Jun 23 17:51:14 on console
WHITEDRAGON:~ markchambers$
```

As you might guess, this text details the last time that you logged into the Terminal. The last line, however, is the more important one. It's called the *prompt.*

The prompt serves some important functions. First, it lists the current directory, which is listed as ~ in the above example. A tilde character (~) denotes a user's Home directory. By default, you're always in your Home folder each time you begin a new session on the Terminal. After the current directory, the Terminal displays the name of the current user, which is `markchambers` in this example.

The final character of the prompt is a $. Consider this your cue because immediately after this character is where you enter any command that you wish to execute. Go ahead; don't be shy. Try out your first command by typing **uptime** in the Terminal application. (It's a good idea to type UNIX commands in lowercase.) Your text appears at the location of the cursor, denoted by a small square. If you make a mistake while entering the command, press the Delete key to back up and then type the characters again. (If the typing error is stuck deep in a longer command, press the left- or right-arrow key to move the cursor immediately after the incorrect character and press Delete to back up; then type the correct characters.) After you type the command, press Return to execute it.

```
WHITEDRAGON:~ markchambers$ uptime
6:24PM  up  2:42, 4 users, load averages: 2.44, 2.38, 1.90
WHITEDRAGON:~ markchambers$
```

If all goes well, you should see a listing of how long your Mac has been running since the last reboot or login. In the example listing, the computer has been running for 2 hours and 42 minutes (`2:42` in line 2). Simple, eh? Immediately following the listing of the `uptime` command, the Terminal displays another prompt for you to enter more commands. I examine many more commands later in this chapter.

Prefer a different appearance for the Terminal window? Click the Terminal menu and choose Preferences; then click the Settings toolbar button to choose the color combinations for the Terminal window background and text.

A few commands to get started

Using the command line is simply a matter of entering simple instructions — or *commands* — into the Terminal application and pressing Return to execute them. It's easy to use the command line to navigate through the various folders on your hard drive. You'll become accustomed to using two vital commands: `ls` and `cd`. The `ls` command is shorthand for *list,* and it does just that: It lists the contents of the current directory. Enter **ls** at your prompt, and you should see a listing of your Home folder.

The complementary `cd` command (lowercase) — which incidentally stands for *change directory* — opens any folder that you specify. It works much the same as double-clicking a folder in the Finder: The difference is that following the `cd` command, you don't immediately see all the folder's content. However, the `cd` command requires a *parameter* (extra options or information that appear after the command) so that your Mac knows which folder to open.

For example, to open the Documents folder that resides in your Home directory, type **cd Documents** and press Return. When you do, you might be surprised to see another prompt displayed immediately. So where are all the files in the Documents folder? You must enter another command to see what items are in the folder that you just opened. Type **ls** again to see the contents of the Documents folder.

If you try to open a folder that has a space in its name, make sure to enclose the folder's name in quotation marks, like this:

```
cd "My Picture Folder"
```

Read more about using quotation marks in your commands in the upcoming section "Command-line gotchas." You can also precede a space in a name with a backslash, like this:

```
cd My\ Picture\ Folder
```

To return to your Home folder in this example, enter a modified version of the `cd` command:

```
cd ..
```

This causes your Mac to move back up the folder hierarchy one folder to your Home directory. By using these three simple commands — `ls`, `cd` *`foldername`*, and `cd ..` — you can traverse your entire hard drive.

After you successfully enter a command, you can recall it by pressing the up-arrow key. Press the up-arrow key again to see the command prior to that, and so forth. This is an extremely useful trick for retyping extra long file paths.

Using the skills you already have

Just because the Terminal is text based doesn't mean that it doesn't act like a good Macintosh citizen. All the usual Mac features that you know and love are there for you to use. Copy and Paste functions work as you might expect — but only at the prompt position.

Drag-and-drop is also at your disposal. After you play around with the Terminal for a while, you'll find yourself bored to tears typing the long paths that represent the files on your hard drive. To automatically enter the path of a file or folder to a command, simply drag it to the active Terminal window, as shown in Figure 1-2. The file's full path instantly appears at the location of your cursor. (Thanks, Apple!)

You can even use the mouse while entering commands in the Terminal. Click and drag your mouse over text to select it. From there, you can copy to the Clipboard as you might expect with any other application.

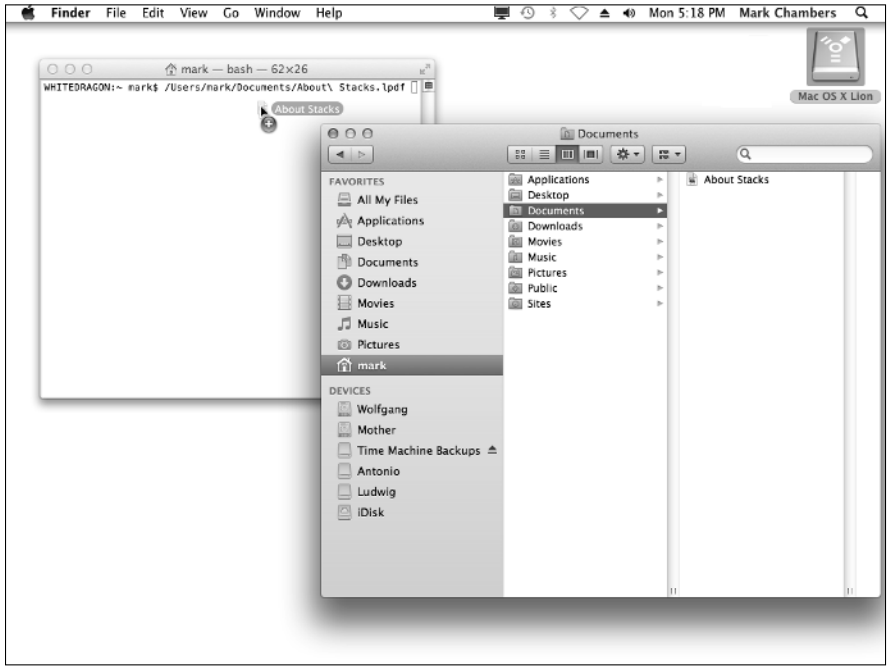

Figure 1-2: Drag a file from a Finder window into Terminal to display its path.

UNIX Commands 101

To use the command line effectively, familiarize yourself with the commands that are available to you. After all, how can you use a tool without knowing what it can do? Despite having to memorize a few commands, UNIX usually makes it easy on you by abbreviating commands, by following a standard grammar (so to speak), and by providing you with extensive documentation for each command.

Anatomy of a UNIX command

UNIX commands can perform many amazing feats. Despite their vast abilities, all commands follow a similar structure:

```
command <optional flag(s)> <optional operand(s)>
```

The simplest form of a UNIX command is the command itself. (For a basic discussion on UNIX commands such as `ls`, see the earlier section, "A few commands to get started.") You can expand your use of the `ls` command by appending various *flags,* which are settings that enable or disable optional features for the command. Most flags are preceded by a dash (-) and always follow the command. For instance, you can display the contents of a directory as a column of names by tacking on a `-l` flag to the `ls` command.

```
ls -l
```

Besides flags, UNIX commands sometimes also have operands. An *operand* is something that is acted upon. For example, instead of just entering the `ls` command, which lists the current directory, you can add an operand to list a specific directory:

```
ls ~/Documents/myProject/
```

The tilde (~) denotes the user's Home directory.

Sometimes a command can take multiple operands, as is the case when you copy a file. The two operands represent the source file and the destination of the file that you want to copy, separated by a space. The following example copies a text file from the Documents folder to the Desktop folder by using the `cp` command (short for *copy*).

```
cp ~/Documents/MyDocument ~/Desktop/MyDocument
```

You can also combine flags and operands in the same command. This example displays the contents of a specific folder in list format:

```
ls -l ~/Documents/myProject/
```

Command-line gotchas

In earlier sections, I describe a few simple command-line functions. All these commands have something in common: You might not have noticed, but every example thus far involved folder names and filenames that contained only alphanumeric characters. Remember what happens if you have a folder name that has a space in it? Try the following example, but don't worry when it doesn't work.

The `cd` command stands for *change directory*.

```
cd /Desktop Folder
```

The result is an error message:

```
-bash: cd: /Desktop: No such file or directory
```

The problem is that a space character isn't allowed in a path. To get around this problem, simply enclose the path in double quotation marks, like this:

```
cd "/Desktop Folder"
```

Mac OS X lets you use either double *or* single quotation marks to enclose a path with spaces in it. Standard UNIX operating systems, however, use double quotation marks for this purpose.

In a similar vein, you can get the space character to be accepted by a command by adding an escape character. To *escape* a character, add a backslash (\) immediately prior to the character in question. To illustrate, try the last command with an escape character instead. Note that this time, no quotation marks are necessary.

```
cd /Desktop\ Folder
```

You can use either quotation marks or escape characters because they're interchangeable.

Help is on the way!

By now, you might be wondering how a computer techno-wizard is supposed to keep all these commands straight. Fortunately, you can find generous documentation for nearly every command available to you. To access this built-in help, use the `man` command. Using the `man` command (shorthand for *manual*) will display a help file for any command that it knows about. For example, to read the available help information for the `ls` command, simply type **man ls** at the prompt. Figure 1-3 illustrates the result.

Autocompletion

To speed things along, the `bash` shell can automagically complete your input for you while you type. (A *shell* takes the commands you type and submits them to the operating system, which then performs the tasks.) Although the Terminal permits you to enter commands via the keyboard, it is the shell that interprets those commands. Many kinds of shells are available to UNIX users. The shell that Lion uses by default is `bash` — another common shell is `tcsh`. Use the autocompletion features of `bash` to autocomplete both commands and filenames. To demonstrate, begin by typing the following:

```
cd ~/De
```

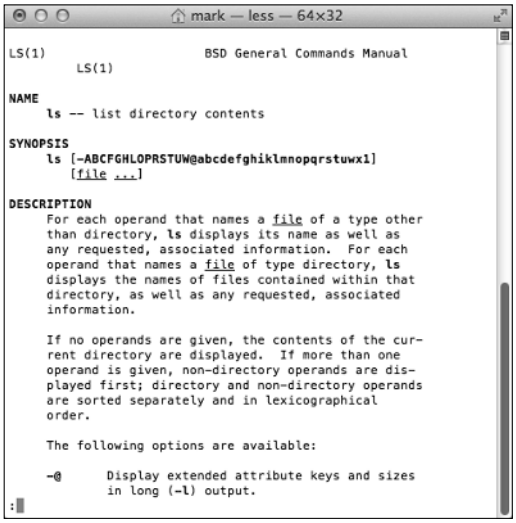
mark — less — 64×32
LS(1) BSD General Commands Manual
LS(1)
NAME
ls -- list directory contents
SYNOPSIS
ls [-ABCFGHLOPRSTUW@abcdefghiklmnopqrstuwx1]
[file ...]
DESCRIPTION
For each operand that names a file of a type other than directory, ls displays its name as well as any requested, associated information. For each operand that names a file of type directory, ls displays the names of files contained within that directory, as well as any requested, associated information.
If no operands are given, the contents of the current directory are displayed. If more than one operand is given, non-directory operands are displayed first; directory and non-directory operands are sorted separately and in lexicographical order.
The following options are available:
-@ Display extended attribute keys and sizes in long (-l) output.

Figure 1-3: Use the man command to display help information.

Then press the Tab key. The result is that the shell predicts that you will want to type

```
cd ~/Desktop/
```

Of course, if you have another folder that begins with the letters *De* in the same folder, you might need to type a few additional characters. This gives the autocompletion feature more information to help it decide which characters you want to type. In other words, if you don't type enough characters, autocompletion ends up like a detective without enough clues to figure things out.

Working with Files

If you've used a computer for any time at all, you're no doubt familiar with the idea of files. Even before the first floppy drive appeared in personal computers, operating systems have stored data in files . . . they date back to the days when a mainframe computer occupied an entire floor of an office building. Mac OS X is no exception, and it's important to understand how Mac OS X arranges them into folders and how you go about accessing them via the command line. This section describes the basic file and folder information that you need to know to tame the beast that is UNIX.

Paths

Before you dive into UNIX commands, you should first know a few facts . . . nasty things, facts, but you can't earn your pair of techno-wizard suspenders without 'em. For starters, as a Mac user, you might not be familiar with how paths work in UNIX. A *path* is simply a textual representation of a folder or file. The simplest path is your Home directory, which is denoted by a tilde character (~) — the tilde character acts as the equivalent of `/Users/<your short account name>` (in my case, `/Users/markchambers`). Any folder within the Home directory is represented by the folder's name preceded by a forward slash (/). For example, a document entitled `myDoc` that resides in the current user's Documents folder would have a path like this:

```
~/Documents/myDoc
```

Similarly, a folder named *myFolder* that resides in the current user's Documents folder would have a path like this:

```
~/Documents/myFolder/
```

As you've probably surmised, a *folder* and a *directory* are two different names for the same thing. *Folder* is the name with which most Mac users are familiar, and *directory* is a term that UNIX power users prefer. I use the terms interchangeably throughout the remainder of the chapter.

Because Mac OS X is a multiuser environment, you might sometimes want to work with folders or files somewhere other than in your Home folder. Starting from your Home folder, enter the following command:

```
cd ..
```

This moves you to the folder right above your Home folder, which happens to be the Users folder. Using another quick `ls` command will show you all users who are permitted to use the machine. (By the way, Shared isn't a user — it's a folder with privileges set so that any user can access its contents.)

Enter **cd ..** once again, and you find yourself at the root of your main hard drive. The *root directory* is what you see in the Finder when you double-click your hard drive icon on the Desktop. A user's Home directory is represented by a tilde character (~), and the root of the hard drive is denoted by a forward slash (/), as displayed by the prompt:

```
WHITEDRAGON:/ markchambers$
```

It's easy to return to your Home directory by following this sequence:

```
WHITEDRAGON:/ markchambers$ cd Users
WHITEDRAGON:/Users markchambers$ cd markchambers
WHITEDRAGON:~ markchambers$
```

Here's a faster way. Instead of moving through each successive folder until you reach your intended destination, you can specify the path by using just one `cd` command:

```
WHITEDRAGON:/ markchambers$ cd /Users/markchambers
WHITEDRAGON:~ markchambers$
```

Of course, the Home directory is a special folder in that you can also navigate there by simply entering **cd ~**, but the main point here is that you can navigate directly to specific folders by using that folder's path in conjunction with the `cd` command.

Furthermore, when you navigate your hard drive by using paths, you can jump directly to your desired destination from any place. When you enter **cd ..**, it is in relation to your current position, whereas entering

```
cd /Users/markchambers
```

always takes you to the same directory, regardless of your starting point.

Copying, moving, renaming, and deleting files

After you're comfortable with moving around the hierarchy of your hard drive, it's a cinch to copy, move, and rename files and folders.

To copy files from the command line, use the `cp` command. Because using the `cp` command will copy a file from one place to another, it requires two operands: first the source and then the destination. For instance, to copy a file from your Home folder to your Documents folder, use the `cp` command like this:

```
cp ~/MyDocument ~/Desktop/MyDocument
```

Keep in mind that when you copy files, **you must have proper permissions to do so!** Here's what happens when I try to copy a file from my Desktop to another user's Desktop (strangely named fuadramses):

```
WHITEDRAGON:~ markchambers$ cp ~/Desktop/MyDocument/Users/fuadramses/Desktop/
    MyDocument
```

Denied! Thwarted! Refused!

```
cp: /Users/fuadramses/Desktop/MyDocument: Permission denied
```

If you can't copy to the destination that you desire, you need to precede the `cp` command with `sudo`. Using the `sudo` command allows you to perform functions as another user. The idea here is that the other user whom you're "emulating" has the necessary privileges to execute the desired copy operation. When you execute the command, the command line asks you for a

password. If you don't know what the password is, you probably shouldn't be using `sudo`. Your computer's administrator should have given you an appropriate password to use. After you enter the correct password, the command executes as desired.

In case you're curious, `sudo` stands for *set user and do*. It sets the user to the one that you specify and performs the command that follows the username.

```
sudo cp ~/Desktop/MyDocument /Users/fuadramses/Desktop/MyDocument
Password:
```

A close cousin to the `cp` (copy) command is the `mv` (move) command. As you can probably guess, the `mv` command moves a folder or file from one location to another. (I told you that all this character-based stuff would start to make sense, didn't I?) To demonstrate, this command moves `MyDocument` from the Desktop folder to the current user's Home folder:

```
mv ~/Desktop/MyDocument ~/MyDocument
```

Ah, but here's the hidden surprise: The `mv` command also functions as a rename command. For instance, to rename a file `MyDocument` on the Desktop to `MyNewDocument`, do this:

```
mv ~/Desktop/MyDocument ~/Desktop/MyNewDocument
```

Because both folders in this example reside in the same folder (`~/Desktop/`), it appears as though the `mv` command has renamed the file.

Again, like the `cp` command, the `mv` command requires that you have proper permissions for the action that you want to perform. Use the `sudo` command to perform any commands that your current user (as displayed in the prompt) isn't allowed to execute. On UNIX systems, not all users are necessarily equal. Some users can perform functions that others can't. This is handy for keeping your child's mitts off important files on your computer. It also creates a hurdle should you choose to work on files while using your child's restricted user account. The `sudo` command lets you temporarily become another user — presumably one that has permission to perform some function that the current user can't.

What would file manipulation be without the ability to delete files? Never fear; UNIX can delete anything that you throw at it. Use the `rm` (short for *remove*) or `rmdir` (short for *remove directory*) command to delete a folder or file. For example, to delete `MyNewDocument` from the Desktop folder, execute the `rm` command like this:

```
rm ~/Desktop/MyNewDocument
```

Once again, deleting files and folders requires that you have permission to do so. In other words, any time that you manipulate files with the command line, you're required to have the proper permission. If your current user lacks these permissions, using `sudo` helps. You should also check to make sure that your target is correctly spelled and that no pesky spaces that could wreak carnage are lurking in the command.

Opening documents and launching applications

Launching applications and opening documents is child's play for a UNIX pro like you. The `open` command does it all. For example, to bring the Finder to the foreground without touching the mouse, use

```
open /System/Library/CoreServices/Finder.app
```

To open a document from the command line, follow a similar scheme. For example, to view an image named `myImage.tif` that's stored in your Documents folder, try this:

```
open ~/Documents/myImage.tif
```

Useful Commands

Manipulating files and viewing folder content is fun, but the command line is capable of so much more! Now I focus your attention on some of the other useful tasks that you can perform with the command line.

Mac OS X comes stocked with a full set of useful commands. You can discover many of the commands that are installed by viewing the files in `/usr/bin`. Type **cd /usr/bin** to navigate there. Other locations to peruse include `/bin` and `/usr/local/bin`.

Calendar

One of my favorite command-line functions is the `cal` command, which displays a calendar in text form. Simply entering **cal** at the prompt displays a calendar for the current month, as shown in Figure 1-4.

Append a number to the `cal` command to display a 12-month calendar for that year. The number that follows the `cal` command is the year for which you'd like to see a calendar. For example, to view a calendar for 1970, type **cal 1970**. The result appears in Figure 1-5.

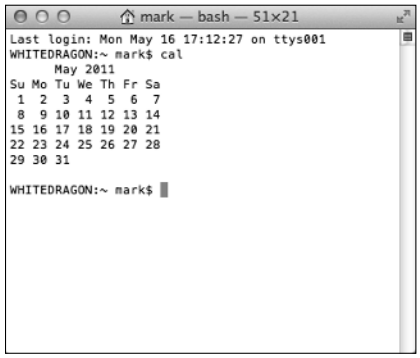
mark — bash — 51×21
Last login: Mon May 16 17:12:27 on ttys001
WHITEDRAGON:~ mark$ cal
May 2011
Su Mo Tu We Th Fr Sa
1 2 3 4 5 6 7
8 9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31
WHITEDRAGON:~ mark$

Figure 1-4: Type **cal** to view a calendar for the current month.

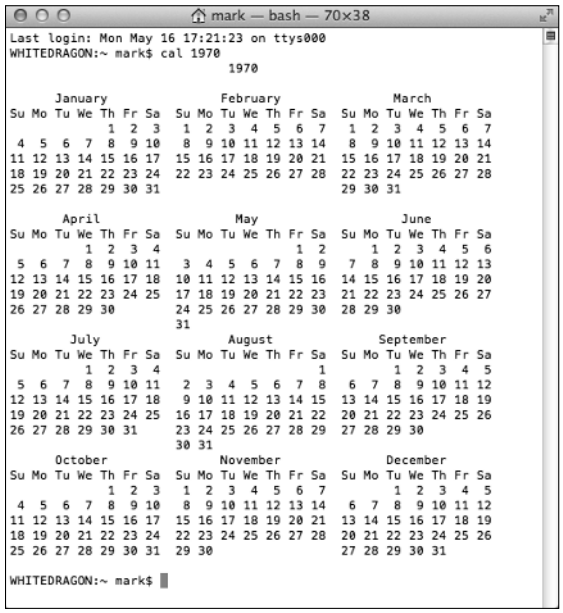

Figure 1-5: Type **cal** followed by a year to view the 12 months of that year.

Append a month number and a year number to display the calendar for that month. For example, to view a calendar for April 2010, type **cal 04 2010**.

Use the `–m` flag to specify the current year, as in **cal –m 08** for August of this year.

Another useful command that's related to the `cal` command is `date`. Type **date** at the command line to display the day, date, time, and year based on your computer's settings.

```
WHITEDRAGON:~ markchambers$ date
Mon Jan 15 11:32:20 CDT 2009
```

Processes

Have you ever been curious as to why your hard drive seems to spin and grind on occasion while your system is seemingly inactive? Mac OS X sometimes has a lot of stuff going on behind the scenes. To discover just what your computer is busy doing at any time, use the `top` command to display all the actions that your computer is currently performing, as shown in Figure 1-6. These activities are called *processes;* some are created when you launch applications, and others are simply tasks that Mac OS X has to take care of to keep things running smoothly.

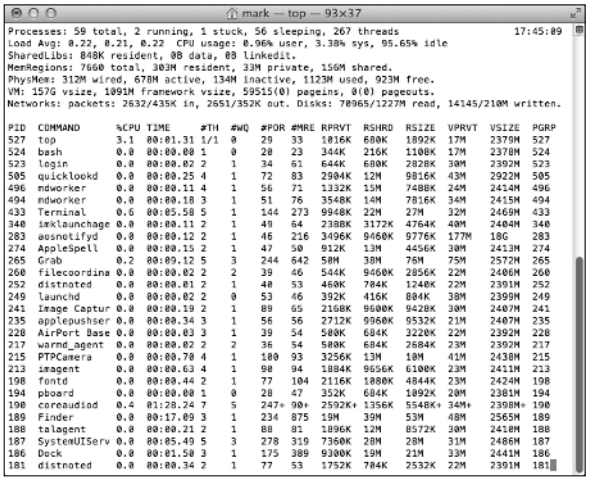

Figure 1-6: The `top` command displays all running processes.

Besides listing the names of the various processes currently in use, `top` tells you how much of your CPU is being devoted to each process. This lets you know what process is currently hogging all your computing power.

Sometimes a process stalls, effectively freezing that action. By using the `top` command to find the Process ID (PID) of the offending process, you can halt the process. Simply use the `kill` command followed by the PID of the process that you want to stop. (The `man` help page for the `kill` command gives more options that may help terminate stubborn processes with prejudice.)

Do *not* go killing processes with a cavalier attitude! Although Mac OS X is extremely stable, removing the wrong process — such as `init` or `mach_init` — is rather like removing a leg from one of those deep-sea drilling platforms: the very *definition* of Not Good. You could lock up your system and lose whatever you're doing in other applications. If you simply want to shut down a misbehaving program, go graphical again (at least for a moment) and use the Force Quit menu command from the Apple menu.

Like `top`, another handy command for examining process info is `ps` (short for *process status*). Most often, you'll want to append a few flags to the `ps` command to get the information that you desire. For example, try the following command:

```
ps -aux
```

The `man` page for `ps` explains what each flag means. (Read more about using the `man` command in the earlier section "Help is on the way!")

UNIX Cadillac Commands

Besides working with files and processes, the command line has all kinds of sophisticated commands. For example, with the command line, you have instant access to a variety of tools for finding files or even stringing together commands.

Finding files

The command line gives you a number of ways to search for files on your hard drive. The two most commonly used commands are `find` and `locate`.

To use `find`, specify a starting point for the search followed by the name of the file or folder that you want to find. For example, to find the Fonts folder that belongs to your user, enter the command like this:

```
find ~/ -name "Fonts"
```

You should see at least one result of the `find` command.

```
/Users/markchambers/Library/Fonts
```

One great feature of the `find` command is that you can look for a file or folder in more than one location. Suppose you want to find a file named `MyDocument` that you know resides either in your Documents folder or on your Desktop. For this kind of search, use the `find` command like this:

```
find ~/Documents ~/Desktop -name "MyDocument"
```

In this example, you are telling the `find` command which folders it should search when looking for the file named `MyDocument`.

Using pipes

Nearly all UNIX commands can take on greater abilities by using a construct called the *pipe.* A pipe (`|`) is represented by that funny little vertical line that shows up when you press Shift+\. The pipe routes data from one command

to another one that follows — for example, many UNIX commands produce large amounts of information that can't all fit on one page. (You might have noticed this behavior when you used the `locate` command.) Joining two commands or functions together with the pipe command is *piping.* To tame the screens full of text, pipe the `find` function to the `less` command. The `less` command provides data one page at a time.

```
find ~/ -name "Fonts" | less
```

When the results fill up one page, the data stops and waits for you to press any key (except the Q key) to continue. When you reach the end of the results, press Q to quit and return to a command-line prompt.

UNIX Programs That Come in Handy

As a Macintosh user, you might be surprised to know that many applications on your hard drive don't reside in one of the typical Applications folders of Mac OS X. These applications, in fact, don't have any graphical user interface like what you're accustomed to. They're accessible only from the command line. The remainder of this chapter covers some of these applications.

Text editors

UNIX has many text-editing applications for use at the command line. Some of the more popular ones include `nano`, `vi`, and `emacs`. Each of these text editors has its pros and cons — and say thanks to the thorough folks at Apple, because all three are included with Lion! For my examples here, however, I use `nano` because it's simple to use and sufficient for our needs.

Creating a new document

To create a text file by using `nano`, simply type **nano** at the command line. The result looks like Figure 1-7.

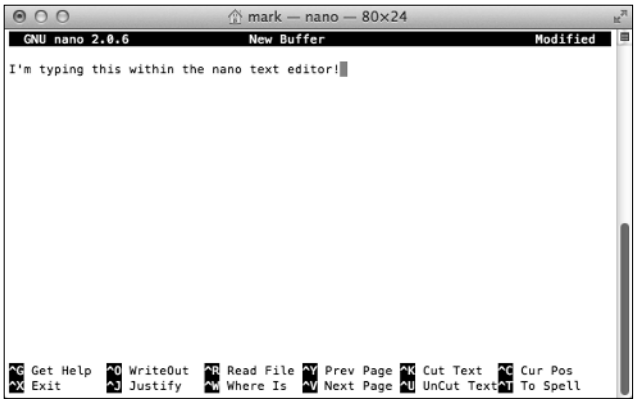

Figure 1-7: The `nano` program is a full-strength text editor, right from the command line.

This is the rough-and-tumble world of UNIX, which preceded the Macintosh by many years. Perhaps this will also help you to appreciate why the Macintosh was so revolutionary when it was introduced. (You can just hear the designers crowing, "We'll call this a *menu!* Yeah, that's the ticket!" The only graphics that you'd see on your monitor were the comics and sticky notes that you stuck to the bottom.)

At the bottom of the screen is a menu of common commands. Above the menu is a large, empty space where you can enter text, much the same as in the word processors that you already know and love. (For those of us that remember the halcyon character-based days of DOS, think older versions of Word and WordPerfect . . . or, if you're a *real* computing dinosaur like I am, consider the original WordStar.) Type some text in that area. Anything will do . . . a letter to a friend, a grocery list, or your school homework.

When you're finished entering your desired text, save the document with the `WriteOut` command in the `nano` menu. Directly next to each command in the `nano` menu is a keyboard sequence used to perform that command. (Refer to the bottom of Figure 1-7.) The ⌥ character is shorthand for the Control key on your keyboard. Thus, to save a file, press ⌥+O. This flies in the face of standard Mac keyboard conventions, where the letter *O* is traditionally used to mean *Open*.

After pressing the Control+O sequence, `pico` prompts you for a filename. As with most UNIX files, you're permitted to enter a simple filename here or a full path to a file. For this example, save the file to your Documents folder, naming it `MyNanoDocument`.

After you've completed and saved the document, pressing Control+X will transport you away from Planet Nano and back to the command line.

Networking with the Terminal

Because UNIX isn't a new phenomenon, it has many useful networking abilities built into it. In fact, UNIX was instrumental in creating much of what we now take for granted: e-mail, the Internet, and the World Wide Web. Thus, you'll be happy to know that you can communicate over networks with the Terminal in practically any manner that you can dream of . . . and then some!

WWW and FTP

If you've used the Internet for any time, you're probably familiar with the various means to transport data over a network. From FTP (short for *File Transfer Protocol*) and Telnet to e-mail and the web, UNIX can handle it all. In fact, UNIX has a command for each of these functions (and many more that have passed into historical obscurity). Rather than use each individual command to send and retrieve data with the Terminal, Apple has conveniently provided a command that can handle them all: `curl`. The `curl` command is

competent at all the standard network protocols. To see it in action, pass a web address (or URL, to The Enlightened) to the `curl` command:

```
curl http://www.mlcbooks.com
```

The result is that you see the HyperText Markup Language (HTML) page that's located at `www.mlcbooks.com`. Because this isn't particularly useful for most people (it's not very easy to read), you need to add the letter `o` as a flag. This specifies where you would like to save this file upon download. To save the HTML page to your Home directory, add the `-o` flag and a path to the destination file.

Don't forget to precede all flags with a hyphen. For this example, it would be `-o`.

```
curl -o ~/mlcbooks.html http://www.mlcbooks.com
```

If you now perform an `ls` command, you see that `curl` has, in fact, downloaded the HTML found at `www.mlcbooks.com` and saved it to a file named `mlcbooks.html` in your Home directory.

The beauty of `curl` is that it does much more than just retrieve web pages: It's equally comfortable with FTP transfers. FTP is used to *download* (or receive) files from a server as well as *upload* (or send) them. Like the previous HyperText Transfer Protocol (HTTP) examples, you only have to provide an FTP address in Uniform Resource Locator (URL) format, and `curl` will take care of the rest. Of course, most people want to save any files that they download via FTP — not view them in the Terminal as I did the HTML file. Therefore, as in the previous example, you should add the `-o` flag and a path to the destination of your download. This time, I download a `README` file about `curl` directly from the makers of `curl`.

```
curl -o ~/Desktop/README.curl ftp://ftp.sunet.se/pub/www/utilities/curl/README.
    curl
```

If you're familiar with FTP, you might be wondering whether `curl` can upload, too. Yes, indeed! Instead of using the `-o` flag, you need to use two flags: `-T` and `-u`. The `-T` flag denotes which file you want to upload. The `-u` flag denotes the username and password. Then, specify the FTP destination address of where you want to upload it. Because this example deals with an upload, the remainder of this example is for an imaginary FTP server. In real life, you'd use the appropriate FTP address, username, and password for an FTP server where you are allowed to upload.

```
curl -T /Desktop/README.curl -u username:passwd ftp://ftp.yoursitehere.com
/myfiles/README.curl
```

This example uploads the `README.curl` file from the Desktop folder that I downloaded earlier to an imaginary FTP server.

How do you spell success? C-u-r-l!

Sure, HTTP and FTP are handy, but did you know that there are many other protocols for network communications? One of the niftier ones is the *Dictionary protocol*. With it, you can look up words from any server that understands the protocol. Suppose, for example, that you want to know the meaning of the term *DVD*. Enter the following command to find out:

```
curl dict://dict.org/d:DVD
```

With `curl`, Dictionary, and your Dictionary Dashboard widget on the same Macintosh, you might never use a "real" paper dictionary again!